C support for IEEE 754-2008

WG14 C floating point study group
Status
24 September 2011
IEEE 754-2008

• Major update to IEEE 754-1985
• 8 Year effort
• Participation by AMD, Apple, HP, IBM, Intel, Sun, academics, etc.
• Adopted as 2011 update to ISO/IEC 60559
• Many new features not supported in C1x nor decimal FP TR 24732
• Less implementation latitude, clearer, bigger than 754-1985
• Recommendations for language standards
• No language binding
• Mostly compatible with C99
Formats (1)

- 128 bit binary quad basic format (recommended)
  - Not required, may be long double
- Unlimited number of interchange formats, which may be arithmetic, including 16 bit binary (optional)
  - Types _FloatN, _DecimalN, N specified in IEC 60559
  - Macros FLT\textunderscore N\_IS\_ARITH, DEC\textunderscore N\_IS\_ARITH defined if arithmetic
  - Names required for supported IEEE types, e.g. _Float32 for float
  - _Float16 required (optionally arithmetic)
  - _FloatN complex, _FloatN imaginary, if _FloatN is arithmetic
Formats (2)

• Extended formats - extending single, double, or quad (recommended)
  - _FloatNx, N= 32, 64, 128
  - _DecimalNx, N = 64, 128
  - Names required for supported qualifying types, e.g., _Float32x for double (if no narrower extended single)
  - Common 80 bit type might be _Float64x

• Extendable formats – user-specified precision and range (recommended)
  - Not planned
Type-related Nomenclature (1)

- Characteristics macros
  - FLT\textit{N\_MAX}, etc.
  - FLT\textit{N\_IS\_ARITH} defined if _Float\textit{N} is arithmetic
  - DE\textit{CN\_MAX}, etc.
  - DE\textit{CN\_IS\_ARITH} defined if _Decimal\textit{N} is arithmetic
  - FLT\textit{NX\_MAX}, etc., for binary extended types
  - DE\textit{CNX\_MAX}, etc., for decimal extended types

- Type classification
  - Non arithmetic interchange types are not floating types
Type Nomeclature (2)

- **Constant suffixes**
  - FN or fN for _FloatN
  - DN or dN for _DecimalN
  - FNx or fNx for _FloatNx
  - DNx or dNx for _DecimalNx

- **Function suffixes**
  - fN for _FloatN
  - dN for _DecimalN
  - fNx for _FloatNx
  - dNx for _DecimalNx
Conversions

• Required for non arithmetic types too
  - Conversions among all interchange types
  - Conversions between character sequences and all interchange types

• Usual arithmetic conversions
  - Convert to wider, wider means exponent range or precision is larger and the other is at least as large
  - Conversion of types not ordered by width is implementation defined
Character sequence conversions

- No new I/O width specifiers
- \texttt{int \texttt{strfromfN}}(char * restrict \texttt{s}, rsize_t \texttt{n}, const char * restrict \texttt{format}, \_Float\texttt{N} \texttt{fp});
- Length modifier inferred from function suffix, not contained in format
- \texttt{strfromf64}(\texttt{s}, \texttt{n}, \texttt{format}, \texttt{fp}) is equivalent \texttt{snprintf}(\texttt{s}, \texttt{n}, \texttt{format}, \texttt{fp})
- \texttt{strfromd\_N}, \texttt{strfromf\_Nx}, \texttt{strfromd\_Nx}
- \texttt{strtof\_N}, \texttt{strtod\_N}, \texttt{strtof\_Nx}, \texttt{strtod\_Nx} added to \texttt{strtod} family

WG 14/N1582
New operations (1)

- **roundToIntegralTiesToEven**
  - `double roundeven(double)`

- **Rounding to integer value with fixed rounding direction must not raise “inexact”**
  - Changed `ceil`, `floor`, `trunc`, and `round` to disallow raising “inexact”

- **nextUp, nextDown**
  - `double nextup(double)`
  - `double nextdown(double)`

- **minNumMag, maxNumMag**
  - `double fminmag(double, double)`
  - `double fmaxmag(double, double)`

WG 14/N1582
New operations (2)

- Integer logb to return outside the range \(\pm 2 \times (\text{emax}+p-1)\) for invalid input
  - long int llogb(double x), FPLLOGB0, FPLLOGBNAN
- formatOf (narrowing) add, sub, mul, div, sqrt, FMA - infinitely precise result rounded to format narrower than parameters
  - float fadd(double x, double y)
  - float faddl(long double x, long double y)
  - double daddl(long double x, long double y)
  - `_FloatM fMaddfN`(_`FloatN` x, _`FloatN` y) for all `M < N`
  - `_FloatM fMaddfNx`(_`FloatNx` x _`FloatfNx`) for all `M <= N` etc.
New operations (3)

- `convertToInteger` functions, with and without inexact signal, for 5 rounding directions (for all integer types, for all floating types???)
  - `intmax_t fromfp(double x, int round, unsigned int width), width = number of bits, round = one of FE_CEIL, FP_FLOOR, FP_TRUNC, FP_ROUND, FP_ROUND_EVEN, without “inexact”`
  - `fromfp(x), with “inexact”`
  - `ufromfp, ufromfp(x), for unsigned integers`
  - An integral-valued rounding function followed by a cast will handle most needs, e.g., `(uint64_t)ceilf128(x) rounds _Float128 upward to uint64_t`
New operations (4)

• Decimal reencoding functions
  - Types for encoded bits dpdencodingdN_t, bidencodingdN_t
  - dpdencodingdN_t encodedpddN(_DecimalN)
  - _DecimalN decodedpddN(dpdencodingdN_t)
  - bidencodingdN_t encodebiddN(_DecimalN)
  - _DecimalN decodebiddN(bidencodingdN_t)

• compareSignalingEqual, compareSignalingNotEqual
  - iseqsig
  - isnesig
New operations (5)

- isSubnormal
  - issubnormal() generic macro
- isSignaling
  - issignaling() generic macro
- Ten-way class
  - Covered by inquiries fpclassify, signbit, and issignaling
- isCanonical
  - iscanonical() generic macro
- totalOrder, totalOrderMag
  - int totalorder(double, double)
  - int totalordermag(double, double)
- raiseFlags (in IEC 60559 “raise” means set bit)
  - int fesetexcept(int excepts)
- Conformance macros
  - TBD
Character string conversions

• Vs C1x Annex F, increase by at least 3 the number of decimal digits for correct rounding to and from binary floating types
  - Require correct rounding for at least DECIMAL_DIG + 3 decimal digits
  - Applies to strtod, scanf, printf families
NaNs

• Requirements similar to IEEE 754-1985
• Propagation recommendations clarified
• NaN payload defined as integral value represented in NaN significand
• Signaling NaNs considered for removal, but retained
  - double canonicalize(double) returns canonical version of input, tiggers signaling NaN inputs (1 * x)
  - int setpayload(double *res, double pl)
  - int setpayloadsignaling(double *res, double pl)
  - double getpayload(const double *)
  - Signaling NaN macros SNANF, ..., DEC_SNAN32, ..., ok for static initialization
Static rounding attributes

• Set rounding direction for a static scope
• Affects all IEC 60559 operations, including sqrt, fma, strtod, printf

☑ In progress
New math functions (recommended) (1)

• New functions (vs C1x):
  - double exp2m1(double)
  - double exp2m1(double)
  - double exp10(double)
  - double exp10m1(double)
  - double log2p1(double)
  - double log10p1(double)
  - double rsqrt(double)
  - double compound(double, double)
New math functions (recommended) (2)

- `double rootn(double, long int)`
- `double pown(double, long int)`
- `double powr(double, double)`
- `double sinpi(double)`
- `double cospi(double)`
- `double tanpi(double)`
- `double tan2pi(double, double)`
Correctly rounded math functions (recommended)

- Correct rounding for all new functions above plus current C1x functions: exp, expm1, exp2, log, log2, log10, logp1, hypot, pow, sin, cos, tan, asin,acos, atan, atan2, sinh, cosh, tanh, asinh, acosh, atanh

 Reserve names with “cr” prefix and all applicable suffixes, e.g., crexp, crlogf, crsinf128, crsqrtd64, for correctly rounded functions
Preferred quantum exponents

- IEC 60559 specifies preferred quantum exponent for its required decimal operations

Specification of preferred quantum exponent for new decimal math functions
Reduction functions (recommended)

- Operate on vectors
- Sum reductions: sum, dot, sumSquare, sumAbs
  - `double reduc_sum(size_t n, const double p[static n])`
  - `double reduc_sumabs(size_t n, const double p[static n])`
  - `double reduc_sumsquare(size_t n, const double p[static n])`
  - `double reduc_sumprod(size_t n, const double p[static n], const double q[static n])`
Scaled reduction functions (recommended)

- Scaled product reductions scaledProd, scaledProdSum, scaledProdDiff – return an in-range scaled product sp and scale factor sf such that the product is \( sp \times 2^{sf} \)
  - `double scaled_prod(size_t n, const double p[static n], long int * restrict sf)`
  - `double scaled_prodsum(size_t n, const double p[static n], const double q[static n], long int * restrict sf)`
  - `double scaled_proddiff(size_t n, const double p[static n], const double q[static n], long int * restrict sf)`
Default modes

• `defaultModes` function to install default settings for all FP modes
  - Type `femode_t`
  - Macro `FE_DFL_MODE`
  - `int fegetmode(femode_t * modep)`
  - `int fesetmode(const femode_t * modep)`
Alternate exception handling (recommended)

• For sub-exceptions as well as exceptions, e.g. invalid from $\infty - \infty$

• Resuming – raiseNoFlag, mayRaiseFlag, recordException, substitute, substituteXor, abruptUnderflow

• Immediate or delayed – break, throw, goto

☑ Not done
Other (recommended)

- Attributes for evaluation methods
  - Not done
- Attributes to allow/disallow optimizations
  - Not done
- Attribute for reproducible results
  - Not done
- Debugging support
  - Not planned